Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №10**

По дисциплине «СПП»

за 6-й семестр

Выполнил:

студент 3 курса

группы ПО-5

Харкевич Д.А.

Проверил:

Крощенко А.А.

Брест, 2022

**Цель работы:** приобрести практические навыки разработки многооконных приложений на JavaFX для работыс базами данных.

**Вариант:** 2

**Задание:**

На основе БД, разработанной в лабораторной работе №9, реализовать многооконное приложениеклиент, позволяющее выполнять основные операции над таблицей в БД (добавление, удаление, модификацию данных).Основные требования к приложению: *•* Для отображения выбирать таблицу с внешними ключами; *•* Осуществлять вывод основных данных в табличном представлении; *•* При выводе краткого представления записи в таблице (т.е. если выводятся не все поля), пощелчку мышкой на запись осуществлять вывод всех полей в подготовленные компоненты наформе; *•* Для всех полей, представленных внешними ключами, выводить их текстовое представлениеиз связанных таблиц (например, таблица-справочник «Времена года» содержит два поля –идентификатор и название сезона, в связанной таблице «Месяц года» есть внешний ключ натаблицу «Времена года»; в этом случае при выводе таблицы «Месяц года» нужно выводитьназвание сезона, а не его идентификатор); *•* При выводе предусмотреть упорядочивание по столбцу; *•* Реализовать простейший фильтр данных по одному-двум полям; *•* При добавлении новых данных в таблицу использовать дополнительное окно для ввода; *•* При модификации данных можно использовать ту же форму, что и для добавления, но с внесенными актуальными значениями полей; *•* При добавлении/модификации выводить варианты значений полей с внешним ключом с помощью выпадающего списка; *•* При удалении данных осуществлять удаление записи, на которой в данных момент находитсяфокус.

**Код программы:**

***Actions***

**InfoDialog**

public class InfoDialog {  
 public InfoDialog(Stage primaryStage, String groupname, String subjectname,  
 Integer semesterid, Integer weekday, String lessonTime, String lecturerFullName){  
 StackPane secondaryLayout = new StackPane();  
  
 final VBox vbox = new VBox();  
  
 vbox.setSpacing(5);  
  
 Label groupLabel = new Label("Group name: " + groupname);  
 Label subjectLabel = new Label("Subject name: " + subjectname);  
 Label semestrLabel = new Label("Semester id: " + semesterid);  
 Label weekLabel = new Label("Week day: " + weekday);  
 Label lessonLabel = new Label("Lesson time: " + lessonTime);  
 Label lecturerNameLabel = new Label("Lecturer name: " + lecturerFullName);  
  
 vbox.getChildren().addAll(groupLabel, subjectLabel, semestrLabel, weekLabel, lessonLabel, lecturerNameLabel);  
 secondaryLayout.getChildren().addAll(vbox);  
  
 Scene secondScene = new Scene(secondaryLayout, 250, 150);  
  
 // New window (Stage)  
 Stage newWindow = new Stage();  
 newWindow.setTitle("Information");  
 newWindow.setScene(secondScene);  
  
 // Specifies the modality for new window.  
 newWindow.initModality(Modality.*WINDOW\_MODAL*);  
  
 // Specifies the owner Window (parent) for new window  
 newWindow.initOwner(primaryStage);  
  
 // Set position of second window, related to primary window.  
 newWindow.setX(primaryStage.getX() + 200);  
 newWindow.setY(primaryStage.getY() + 100);  
  
 newWindow.setHeight(200);  
 newWindow.setWidth(300);  
  
 newWindow.setResizable(false);  
  
 newWindow.show();  
 }  
}

**TimeTableAddingDialog**

public class TimeTableAddingDialog {  
 Label tittleLabel;  
 Button addButton;  
  
 Label weekDayLabel;  
 TextField weekDayField;  
  
 ComboBox<Subject> subjectComboBox;  
 ComboBox<Group> groupComboBox;  
 ComboBox<Lecturer> lecturerComboBox;  
 ComboBox<Calendar> calendarComboBox;  
  
 public TimeTableAddingDialog(Stage primaryStage) {  
 initControls();  
  
 List<Subject> subjects;  
 List<Group> groups;  
 List<Lecturer> lecturers;  
 List<Calendar> calendars;  
  
 try {  
 DBManager dbManager = new DBManager();  
  
 subjects = dbManager.*subjectTable*.GetAll();  
 groups = dbManager.*groupTable*.GetAll();  
 lecturers = dbManager.*lecturerTable*.GetAll();  
 calendars = dbManager.*calendarTable*.GetAll();  
  
 ObservableList<Subject> subjectObservableList = FXCollections.*observableArrayList*(subjects);  
 ObservableList<Group> groupObservableList = FXCollections.*observableArrayList*(groups);  
 ObservableList<Lecturer> lecturerObservableList = FXCollections.*observableArrayList*(lecturers);  
 ObservableList<Calendar> calendarObservableList = FXCollections.*observableArrayList*(calendars);  
  
 subjectComboBox = new ComboBox<>(subjectObservableList);  
 groupComboBox = new ComboBox<>(groupObservableList);  
 lecturerComboBox = new ComboBox<>(lecturerObservableList);  
 calendarComboBox = new ComboBox<>(calendarObservableList);  
  
 subjectComboBox.setValue(subjectObservableList.get(0));  
 groupComboBox.setValue(groupObservableList.get(0));  
 lecturerComboBox.setValue(lecturerObservableList.get(0));  
 calendarComboBox.setValue(calendarObservableList.get(0));  
 }  
 catch (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
  
 StackPane secondaryLayout = new StackPane();  
 final VBox vbox = new VBox();  
  
 vbox.setSpacing(5);  
  
 vbox.getChildren().addAll(tittleLabel, weekDayLabel, weekDayField,  
 new Label("Enter subject"), subjectComboBox,  
 new Label("Enter group"), groupComboBox,  
 new Label("Enter lecturer"), lecturerComboBox,  
 new Label("Enter lesson"), calendarComboBox, addButton);  
 secondaryLayout.getChildren().addAll(vbox);  
  
 Scene secondScene = new Scene(secondaryLayout, 300, 240);  
  
 // New window (Stage)  
 Stage newWindow = new Stage();  
 newWindow.setTitle("Add new information");  
 newWindow.setScene(secondScene);  
  
 // Specifies the modality for new window.  
 newWindow.initModality(Modality.*WINDOW\_MODAL*);  
  
 // Specifies the owner Window (parent) for new window  
 newWindow.initOwner(primaryStage);  
  
 // Set position of second window, related to primary window.  
 newWindow.setX(primaryStage.getX() + 200);  
 newWindow.setY(primaryStage.getY() + 200);  
  
 newWindow.setHeight(400);  
 newWindow.setWidth(250);  
 newWindow.setResizable(false);  
  
 newWindow.show();  
  
 addButton.setOnAction(event -> {  
 Integer weekDay = Integer.*parseInt*(weekDayField.getText());  
  
 Integer subjectId = subjectComboBox.getSelectionModel().getSelectedItem().getId();  
 Integer groupId = groupComboBox.getSelectionModel().getSelectedItem().getId();  
 Integer lecturerId = lecturerComboBox.getSelectionModel().getSelectedItem().getId();  
 Integer calendarId = calendarComboBox.getSelectionModel().getSelectedItem().getId();  
  
 if (weekDay > 7 || weekDay <= 0)  
 return;  
  
 try {  
 DBManager repositoryManager = new DBManager();  
 repositoryManager.*scheduleTable*.Add(new TimeTable(groupId, subjectId, lecturerId, weekDay, calendarId));  
 newWindow.close();  
 }  
 catch (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 });  
 }  
  
 public void initControls()  
 {  
 tittleLabel = new Label("Enter the information:");  
  
 weekDayLabel = new Label("Week day");  
 weekDayField = new TextField();  
  
 addButton = new Button("Add the information");  
  
 weekDayField.setMaxSize(50, 50);  
 }  
}

**TimeTableUpdatingDialog**

public class TimeTableUpdatingDialog {  
 Label tittleLabel;  
 Button addButton;  
  
 Label weekDayLabel;  
 TextField weekDayField;  
  
 ComboBox<Subject> subjectComboBox;  
 ComboBox<Group> groupComboBox;  
 ComboBox<Lecturer> lecturerComboBox;  
 ComboBox<Calendar> calendarComboBox;  
  
 public TimeTableUpdatingDialog(Stage primaryStage, TimeTable table) {  
 initControls();  
  
 List<Subject> subjects;  
 List<Group> groups;  
 List<Lecturer> lecturers;  
 List<Calendar> calendars;  
  
 try {  
 DBManager dbManager = new DBManager();  
  
 subjects = dbManager.*subjectTable*.GetAll();  
 groups = dbManager.*groupTable*.GetAll();  
 lecturers = dbManager.*lecturerTable*.GetAll();  
 calendars = dbManager.*calendarTable*.GetAll();  
  
 ObservableList<Subject> subjectObservableList = FXCollections.*observableArrayList*(subjects);  
 ObservableList<Group> groupObservableList = FXCollections.*observableArrayList*(groups);  
 ObservableList<Lecturer> lecturerObservableList = FXCollections.*observableArrayList*(lecturers);  
 ObservableList<Calendar> calendarObservableList = FXCollections.*observableArrayList*(calendars);  
  
 subjectComboBox = new ComboBox<>(subjectObservableList);  
 groupComboBox = new ComboBox<>(groupObservableList);  
 lecturerComboBox = new ComboBox<>(lecturerObservableList);  
 calendarComboBox = new ComboBox<>(calendarObservableList);  
  
 subjectComboBox.setValue(subjectObservableList.get(table.getSubjectid() - 1));  
 groupComboBox.setValue(groupObservableList.get(table.getGroupid() - 1));  
 lecturerComboBox.setValue(lecturerObservableList.get(table.getLecturerid()));  
 calendarComboBox.setValue(calendarObservableList.get(table.getLessonid() - 1));  
  
 String text = Integer.*toString*(table.getWeekday()) ;  
 weekDayField.setText(text);  
 }  
 catch (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
  
 StackPane secondaryLayout = new StackPane();  
 final VBox vbox = new VBox();  
  
 vbox.setSpacing(5);  
  
 vbox.getChildren().addAll(tittleLabel, weekDayLabel, weekDayField,  
 new Label("Enter subject"), subjectComboBox,  
 new Label("Enter group"), groupComboBox,  
 new Label("Enter lecturer"), lecturerComboBox,  
 new Label("Enter lesson"), calendarComboBox, addButton);  
 secondaryLayout.getChildren().addAll(vbox);  
  
 Scene secondScene = new Scene(secondaryLayout, 300, 240);  
  
 // New window (Stage)  
 Stage newWindow = new Stage();  
 newWindow.setTitle("Update cell");  
 newWindow.setScene(secondScene);  
  
 // Specifies the modality for new window.  
 newWindow.initModality(Modality.*WINDOW\_MODAL*);  
  
 // Specifies the owner Window (parent) for new window  
 newWindow.initOwner(primaryStage);  
  
 // Set position of second window, related to primary window.  
 newWindow.setX(primaryStage.getX() + 200);  
 newWindow.setY(primaryStage.getY() + 200);  
  
 newWindow.setHeight(400);  
 newWindow.setWidth(250);  
 newWindow.setResizable(false);  
  
 newWindow.show();  
  
 addButton.setOnAction(event -> {  
 Integer id = table.getId();  
  
 Integer weekDay = Integer.*parseInt*(weekDayField.getText());  
 Integer subjectId = subjectComboBox.getSelectionModel().getSelectedItem().getId();  
 Integer groupId = groupComboBox.getSelectionModel().getSelectedItem().getId();  
 Integer lecturerId = lecturerComboBox.getSelectionModel().getSelectedItem().getId();  
 Integer calendarId = calendarComboBox.getSelectionModel().getSelectedItem().getId();  
  
 if (weekDay > 7 || weekDay < 0)  
 return;  
  
 try {  
 DBManager dbManager = new DBManager();  
 dbManager.*scheduleTable*.Update(new TimeTable(id, groupId,  
 subjectId, lecturerId, weekDay, calendarId));  
 newWindow.close();  
 }  
 catch (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 });  
 }  
  
 public void initControls()  
 {  
 tittleLabel = new Label("Enter the information:");  
  
 weekDayLabel = new Label("Week day");  
 weekDayField = new TextField();  
  
 addButton = new Button("Update cell");  
  
 weekDayField.setMaxSize(50, 50);  
 }  
}

***Database***

**Connection**

public final class Connection {  
  
 public java.sql.Connection GetConnection() throws SQLException {  
 String connectionURL = "jdbc:mysql://localhost:3306/mysql";  
 String connectionUser = "root";  
 String connectionPassword = "root";  
  
 java.sql.Connection connection = DriverManager.*getConnection*(connectionURL, connectionUser, connectionPassword);  
 return connection;  
 }  
}

**DBManager**

public class DBManager {  
 public static DBInterface<Subject> *subjectTable*;  
 public static DBInterface<Group> *groupTable*;  
 public static DBInterface<Lecturer> *lecturerTable*;  
 public static DBInterface<Calendar> *calendarTable*;  
 public static DBInterface<TimeTable> *scheduleTable*;  
  
 private static final Connection *connection* = new Connection();  
  
 public DBManager() throws SQLException {  
 this.*subjectTable* = new SubjectTable(*connection*.GetConnection());  
 this.*groupTable* = new GroupTable(*connection*.GetConnection());  
 this.*lecturerTable* = new LecturerTable(*connection*.GetConnection());  
 this.*calendarTable* = new CalendarTable(*connection*.GetConnection());  
 this.*scheduleTable* = new ScheduleTable(*connection*.GetConnection());  
 }   
}

**DBInterface**

public interface DBInterface <T> {  
 T Add(T entity) throws SQLException;  
 void Update(T entity) throws SQLException;  
 void Delete(T entity) throws SQLException;  
 T GetByIdOrNull(int id) throws SQLException;  
 ArrayList<T> GetAll() throws SQLException;  
}

**CalendarTable**

public class CalendarTable implements DBInterface<Calendar> {  
  
 private final Connection connection;  
  
 public CalendarTable(Connection connection) { this.connection = connection; }  
  
 @Override  
 public Calendar Add(Calendar entity) throws SQLException {  
 String query = "INSERT INTO schedule.calendar( " +  
 " semesterid, weekday, lessonid, lessontime) " +  
 " VALUES (?, ?, ?, ?)";  
  
 PreparedStatement statement = connection.prepareStatement(query, Statement.*RETURN\_GENERATED\_KEYS*);  
  
 statement.setInt(1, entity.getSemesterid());  
 statement.setInt(2, entity.getWeekday());  
 statement.setInt(3, entity.getLessonid());  
 statement.setString(4, entity.getLessontime());  
  
 statement.execute();  
  
 ResultSet generatedKeys = statement.getGeneratedKeys();  
 generatedKeys.next();  
 entity.setId(generatedKeys.getInt(1));  
  
 return entity;  
 }  
  
 @Override  
 public void Update(Calendar entity) throws SQLException {  
 String query ="UPDATE schedule.calendar " +  
 " SET semesterid=?, weekday=?, lessonid=?, lessontime=?" +  
 " WHERE id=?";  
  
 PreparedStatement statement = connection.prepareStatement(query);  
 statement.setInt(1, entity.getSemesterid());  
 statement.setInt(2, entity.getWeekday());  
 statement.setInt(3, entity.getLessonid());  
 statement.setString(4, entity.getLessontime());  
 statement.setInt(5, entity.getId());  
  
 statement.executeUpdate();  
 }  
  
 @Override  
 public void Delete(Calendar entity) throws SQLException {  
 String query = "DELETE FROM schedule.calendar" +  
 " WHERE id=?";  
 PreparedStatement statement = connection.prepareStatement(query);  
 statement.setInt(1, entity.getId());  
  
 statement.executeUpdate();  
 }  
  
 @Override  
 public Calendar GetByIdOrNull(int id) throws SQLException {  
 String query =  
 "SELECT \* FROM schedule.calendar" +  
 " WHERE Id = ?";  
  
 PreparedStatement statement = connection.prepareStatement(query);  
 statement.setInt(1, id);  
  
 ResultSet reader = statement.executeQuery();  
 if(reader.next())  
 {  
 Calendar result = new Calendar();  
 result.setId(reader.getInt("id"));  
 result.setLessonid(reader.getInt("lessonid"));  
 result.setSemesterid(reader.getInt("semesterid"));  
 result.setWeekday(reader.getInt("weekday"));  
 result.setLessontime(reader.getString("lessontime"));  
 return result;  
 }  
  
 return null;  
 }  
  
 @Override  
 public ArrayList<Calendar> GetAll() throws SQLException {  
 String query =  
 "SELECT \* FROM schedule.calendar Order by id";  
  
 PreparedStatement statement = connection.prepareStatement(query);  
  
 ResultSet reader = statement.executeQuery();  
 ArrayList result = new ArrayList<Calendar>();  
 while (reader.next())  
 {  
 Calendar calendar = new Calendar();  
 calendar.setId(reader.getInt("id"));  
 calendar.setLessonid(reader.getInt("lessonid"));  
 calendar.setSemesterid(reader.getInt("semesterid"));  
 calendar.setWeekday(reader.getInt("weekday"));  
 calendar.setLessontime(reader.getString("lessontime"));  
  
 result.add(calendar);  
 }  
  
 return result;  
 }  
}

**GroupTable**

public class GroupTable implements DBInterface<Group> {  
  
 private final Connection connection;  
  
 public GroupTable(Connection connection) {  
 this.connection = connection;  
 }  
  
 @Override  
 public Group Add(Group entity) throws SQLException {  
 String query =  
 "INSERT INTO schedule.groupsu(" +  
 " groupname)" +  
 " VALUES (?)";  
  
 PreparedStatement statement = connection.prepareStatement(query, Statement.*RETURN\_GENERATED\_KEYS*);  
 statement.setString(1, entity.getGroupname());  
  
 statement.execute();  
  
 ResultSet generatedKeys = statement.getGeneratedKeys();  
 generatedKeys.next();  
 entity.setId(generatedKeys.getInt(1));  
  
 return entity;  
 }  
  
 @Override  
 public void Update(Group entity) throws SQLException {  
 String query =  
 "UPDATE schedule.groupsu" +  
 " SET groupname = ?" +  
 " WHERE id = ?";  
  
 PreparedStatement statement = connection.prepareStatement(query);  
 statement.setString(1, entity.getGroupname());  
 statement.setInt(2, entity.getId());  
  
 statement.executeUpdate();  
 }  
  
 @Override  
 public void Delete(Group entity) throws SQLException {  
 String query = "DELETE FROM schedule.groupsu" +  
 " WHERE id=?";  
 PreparedStatement statement = connection.prepareStatement(query);  
 statement.setInt(1, entity.getId());  
  
 statement.executeUpdate();  
 }  
  
 @Override  
 public Group GetByIdOrNull(int id) throws SQLException {  
 String query =  
 "SELECT \* FROM schedule.groupsu" +  
 " WHERE Id = ?";  
  
 PreparedStatement statement = connection.prepareStatement(query);  
 statement.setInt(1, id);  
  
 ResultSet reader = statement.executeQuery();  
 if(reader.next())  
 {  
 Group result = new Group();  
 result.setId(reader.getInt("id"));  
 result.setGroupname(reader.getString("groupname"));  
 return result;  
 }  
  
 return null;  
 }  
  
 @Override  
 public ArrayList<Group> GetAll() throws SQLException {  
 String query =  
 "SELECT \* FROM schedule.groupsu Order by id";  
  
 PreparedStatement statement = connection.prepareStatement(query);  
  
 ResultSet reader = statement.executeQuery();  
 ArrayList result = new ArrayList<Group>();  
 while (reader.next())  
 {  
 Group group = new Group();  
 group.setId(reader.getInt("id"));  
 group.setGroupname(reader.getString("groupname"));  
  
 result.add(group);  
 }  
  
 return result;  
 }  
}

**LecturerTable**

public class LecturerTable implements DBInterface<Lecturer> {  
  
 private final Connection connection;  
  
 public LecturerTable(Connection connection) {  
 this.connection = connection;  
 }  
  
 @Override  
 public Lecturer Add(Lecturer entity) throws SQLException {  
 String query =  
 "INSERT INTO schedule.lecturers(" +  
 "firstname, lastname, patronymic)" +  
 " VALUES (?, ?, ?)";  
  
 PreparedStatement statement = connection.prepareStatement(query, Statement.*RETURN\_GENERATED\_KEYS*);  
 statement.setString(1, entity.getFirstname());  
 statement.setString(2, entity.getLastname());  
 statement.setString(3, entity.getPatronymic());  
  
 statement.execute();  
  
 ResultSet generatedKeys = statement.getGeneratedKeys();  
 generatedKeys.next();  
 entity.setId(generatedKeys.getInt(1));  
  
 return entity;  
 }  
  
 @Override  
 public void Update(Lecturer entity) throws SQLException {  
 String query =  
 "UPDATE schedule.lecturers" +  
 " SET firstname=?, lastname=?, patronymic=?" +  
 " WHERE id=?";  
  
  
 PreparedStatement statement = connection.prepareStatement(query);  
 statement.setString(1, entity.getFirstname());  
 statement.setString(2, entity.getLastname());  
 statement.setString(3, entity.getPatronymic());  
 statement.setInt(4, entity.getId());  
  
 statement.executeUpdate();  
 }  
  
 @Override  
 public void Delete(Lecturer entity) throws SQLException {  
 String query = "DELETE FROM schedule.lecturers" +  
 " WHERE id=?";  
 PreparedStatement statement = connection.prepareStatement(query);  
 statement.setInt(1, entity.getId());  
  
 statement.executeUpdate();  
 }  
  
 @Override  
 public Lecturer GetByIdOrNull(int id) throws SQLException {  
 String query =  
 "SELECT \* FROM schedule.lecturers" +  
 " WHERE Id = ?";  
  
 PreparedStatement statement = connection.prepareStatement(query);  
 statement.setInt(1, id);  
  
 ResultSet reader = statement.executeQuery();  
 if(reader.next())  
 {  
 Lecturer result = new Lecturer();  
 result.setId(reader.getInt("id"));  
 result.setFirstname(reader.getString("firstname"));  
 result.setLastname(reader.getString("lastname"));  
 result.setPatronymic(reader.getString("patronymic"));  
 return result;  
 }  
  
 return null;  
 }  
  
 @Override  
 public ArrayList<Lecturer> GetAll() throws SQLException {  
 String query =  
 "SELECT \* FROM schedule.lecturers Order by id";  
  
 PreparedStatement statement = connection.prepareStatement(query);  
  
 ResultSet reader = statement.executeQuery();  
 ArrayList result = new ArrayList<Lecturer>();  
 while (reader.next())  
 {  
 Lecturer lecturer = new Lecturer();  
 lecturer.setId(reader.getInt("id"));  
 lecturer.setFirstname(reader.getString("firstname"));  
 lecturer.setLastname(reader.getString("lastname"));  
 lecturer.setPatronymic(reader.getString("patronymic"));  
  
 result.add(lecturer);  
 }  
  
 return result;  
 }  
  
}

**ScheduleTable**

public class ScheduleTable implements DBInterface<TimeTable> {  
 private final Connection connection;  
  
 public ScheduleTable(Connection connection) {  
 this.connection = connection;  
 }  
  
 @Override  
 public TimeTable Add(TimeTable entity) throws SQLException {  
 String query =  
 "INSERT INTO schedule.timetable(" +  
 "groupid, subjectid, lecturerid, weekday, lessonid)" +  
 " VALUES (?, ?, ?, ?, ?);";  
  
 PreparedStatement statement = connection.prepareStatement(query, Statement.*RETURN\_GENERATED\_KEYS*);  
 statement.setInt(1, entity.getGroupid());  
 statement.setInt(2, entity.getSubjectid());  
 statement.setInt(3, entity.getLecturerid());  
 statement.setInt(4, entity.getWeekday());  
 statement.setInt(5, entity.getLessonid());  
  
 statement.execute();  
  
 ResultSet generatedKeys = statement.getGeneratedKeys();  
 generatedKeys.next();  
 entity.setId(generatedKeys.getInt(1));  
  
 return entity;  
 }  
  
 @Override  
 public void Update(TimeTable entity) throws SQLException {  
  
 String query = "UPDATE schedule.timetable " +  
 " SET groupid=?, subjectid=?, lecturerid=?, weekday=?, lessonid=? " +  
 " WHERE id=?";  
  
 PreparedStatement statement = connection.prepareStatement(query);  
 statement.setInt(1, entity.getGroupid());  
 statement.setInt(2, entity.getSubjectid());  
 statement.setInt(3, entity.getLecturerid());  
 statement.setInt(4, entity.getWeekday());  
 statement.setInt(5, entity.getLessonid());  
 statement.setInt(6, entity.getId());  
  
 statement.executeUpdate();  
 return;  
 }  
  
 @Override  
 public void Delete(TimeTable entity) throws SQLException {  
 String query = "DELETE FROM schedule.timetable" +  
 " WHERE id=?";  
  
 PreparedStatement statement = connection.prepareStatement(query);  
 statement.setInt(1, entity.getId());  
  
 statement.executeUpdate();  
 }  
  
 @Override  
 public TimeTable GetByIdOrNull(int id) throws SQLException {  
 String query =  
 "SELECT \* FROM schedule.timetable" +  
 " WHERE Id = ?";  
  
 PreparedStatement statement = connection.prepareStatement(query);  
 statement.setInt(1, id);  
  
 ResultSet reader = statement.executeQuery();  
 if(reader.next())  
 {  
 TimeTable result = new TimeTable();  
 result.setId(reader.getInt("id"));  
 result.setLessonid(reader.getInt("lessonid"));  
 result.setWeekday(reader.getInt("weekday"));  
 result.setGroupid(reader.getInt("groupid"));  
 result.setSubjectid(reader.getInt("subjectid"));  
 return result;  
 }  
  
 return null;  
 }  
  
 @Override  
 public ArrayList<TimeTable> GetAll() throws SQLException {  
 String query =  
 "SELECT \* FROM schedule.timetable Order by id";  
  
 PreparedStatement statement = connection.prepareStatement(query);  
  
 ResultSet reader = statement.executeQuery();  
 ArrayList result = new ArrayList<TimeTable>();  
 while (reader.next())  
 {  
 TimeTable timeTable = new TimeTable();  
 timeTable.setId(reader.getInt("id"));  
 timeTable.setGroupid(reader.getInt("groupid"));  
 timeTable.setSubjectid(reader.getInt("subjectid"));  
 timeTable.setLecturerid(reader.getInt("lecturerid"));  
 timeTable.setWeekday(reader.getInt("weekday"));  
 timeTable.setLessonid(reader.getInt("lessonid"));  
  
 result.add(timeTable);  
 }  
  
 return result;  
 }  
}

**SubjectTable**

public class SubjectTable implements DBInterface<Subject> {  
 private final Connection connection;  
  
 public SubjectTable(Connection connection) {  
 this.connection = connection;  
 }  
  
 @Override  
 public Subject Add(Subject entity) throws SQLException {  
 String query =  
 "INSERT INTO schedule.subjects(" +  
 " subjectname)" +  
 " VALUES (?)";  
  
 PreparedStatement statement = connection.prepareStatement(query, Statement.*RETURN\_GENERATED\_KEYS*);  
 statement.setString(1, entity.getSubjectName());  
  
 statement.execute();  
  
 ResultSet generatedKeys = statement.getGeneratedKeys();  
 generatedKeys.next();  
 entity.setId(generatedKeys.getInt(1));  
  
 return entity;  
 }  
  
 @Override  
 public void Update(Subject entity) throws SQLException {  
 String query =  
 "UPDATE schedule.subjects" +  
 " SET subjectname = ?" +  
 " WHERE id = ?";  
  
 PreparedStatement statement = connection.prepareStatement(query);  
 statement.setString(1, entity.getSubjectName());  
 statement.setInt(2, entity.getId());  
  
 statement.executeUpdate();  
 }  
  
 @Override  
 public void Delete(Subject entity) throws SQLException {  
 String query = "DELETE FROM schedule.subjects" +  
 " WHERE id=?";  
 PreparedStatement statement = connection.prepareStatement(query);  
 statement.setInt(1, entity.getId());  
  
 statement.executeUpdate();  
 }  
  
 @Override  
 public Subject GetByIdOrNull(int id) throws SQLException {  
 String query =  
 "SELECT \* FROM schedule.subjects" +  
 " WHERE Id = ? " +  
 "Order by id";  
  
 PreparedStatement statement = connection.prepareStatement(query);  
 statement.setInt(1, id);  
  
 ResultSet reader = statement.executeQuery();  
 if(reader.next())  
 {  
 Subject result = new Subject();  
 result.setId(reader.getInt("id"));  
 result.setSubjectName(reader.getString("subjectname"));  
 return result;  
 }  
  
 return null;  
 }  
  
 @Override  
 public ArrayList<Subject> GetAll() throws SQLException {  
 String query =  
 "SELECT \* FROM schedule.subjects Order by id";  
  
 PreparedStatement statement = connection.prepareStatement(query);  
  
 ResultSet reader = statement.executeQuery();  
 ArrayList result = new ArrayList<Subject>();  
 while (reader.next())  
 {  
 Subject subject = new Subject();  
 subject.setId(reader.getInt("id"));  
 subject.setSubjectName(reader.getString("subjectname"));  
  
 result.add(subject);  
 }  
  
 return result;  
 }  
}

***Entities***

**Calendar**

public class Calendar {  
 private int id;  
 private int semesterid;  
 private int weekday;  
 private int lessonid;  
 private String lessontime;  
  
 public Calendar() {}  
  
 public Calendar(int semesterId, int weekDay, int lessonId, String lessonTime) {  
 this.semesterid = semesterId;  
 this.weekday = weekDay;  
 this.lessonid = lessonId;  
 this.lessontime = lessonTime;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public int getSemesterid() {  
 return semesterid;  
 }  
  
 public void setSemesterid(int semesterId) {  
 this.semesterid = semesterId;  
 }  
  
 public int getWeekday() {  
 return weekday;  
 }  
  
 public void setWeekday(int weekDay) {  
 this.weekday = weekDay;  
 }  
  
 public int getLessonid() {  
 return lessonid;  
 }  
  
 public void setLessonid(int lessonId) {  
 this.lessonid = lessonId;  
 }  
  
 public String getLessontime() {  
 return lessontime;  
 }  
  
 public void setLessontime(String lessonTime) {  
 this.lessontime = lessonTime;  
 }  
  
 @Override  
 public String toString()  
 {  
 return getLessontime();  
 }  
}

**Group**

public class Group {  
 private int id;  
 private String groupname;  
  
 public Group() {}  
  
 public Group(String groupName) {  
 this.groupname = groupName;  
 }  
  
 public String getGroupname() {  
 return groupname;  
 }  
  
 public void setGroupname(String groupName) {  
 this.groupname = groupName;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 @Override  
 public String toString()  
 {  
 return getGroupname();  
 }  
}

**Lecturer**

public class Lecturer {  
 private int id;  
 private String firstname;  
 private String lastname;  
 private String patronymic;  
  
 public Lecturer() {}  
  
 public Lecturer(String firstName, String lastName, String patronymic) {  
 this.firstname = firstName;  
 this.lastname = lastName;  
 this.patronymic = patronymic;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getFirstname() {  
 return firstname;  
 }  
  
 public void setFirstname(String firstName) {  
 this.firstname = firstName;  
 }  
  
 public String getLastname() {  
 return lastname;  
 }  
  
 public void setLastname(String lastName) {  
 this.lastname = lastName;  
 }  
  
 public String getPatronymic() {  
 return patronymic;  
 }  
  
 public void setPatronymic(String patronymic) {  
 this.patronymic = patronymic;  
 }  
  
 @Override  
 public String toString()  
 {  
 return getFirstname() + " " + getLastname();  
 }  
}

**Subject**

public class Subject {  
 private int id;  
 private String subjectName;  
  
 public Subject() {}  
  
 public Subject(String subjectName) {  
 this.subjectName = subjectName;  
 }  
  
 public String getSubjectName() {  
 return subjectName;  
 }  
  
 public void setSubjectName(String subjectName) {  
 this.subjectName = subjectName;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 @Override  
 public String toString()  
 {  
 return getSubjectName();  
 }  
}

**TimeTable**

public class TimeTable {  
 private int id;  
  
 private int groupid;  
 private Group group;  
  
 private int subjectid;  
 private Subject subject;  
  
 private int lecturerid;  
 private Lecturer lecturer;  
  
 private int weekday;  
  
 private int lessonid;  
 private Calendar calendar;  
  
 public TimeTable() {}  
  
 public TimeTable(int groupId, int subjectId, int lecturerId, int weekDay, int lessonId) {  
 this.groupid = groupId;  
 this.subjectid = subjectId;  
 this.lecturerid = lecturerId;  
 this.weekday = weekDay;  
 this.lessonid = lessonId;  
 }  
  
 public TimeTable(int id) {  
 this.id = id;  
 }  
  
 public TimeTable(int id, int groupId, int subjectId, int lecturerId, int weekDay, int lessonId) {  
 this.id = id;  
 this.groupid = groupId;  
 this.subjectid = subjectId;  
 this.lecturerid = lecturerId;  
 this.weekday = weekDay;  
 this.lessonid = lessonId;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public int getGroupid() {  
 return groupid;  
 }  
  
 public void setGroupid(int groupId) {  
 this.groupid = groupId;  
 }  
  
 public Group getGroup() {  
 return group;  
 }  
  
 public void setGroup(Group group) {  
 this.group = group;  
 }  
  
 public int getSubjectid() {  
 return subjectid;  
 }  
  
 public void setSubjectid(int subjectId) {  
 this.subjectid = subjectId;  
 }  
  
 public Subject getSubject() {  
 return subject;  
 }  
  
 public void setSubject(Subject subject) {  
 this.subject = subject;  
 }  
  
 public int getLecturerid() {  
 return lecturerid;  
 }  
  
 public void setLecturerid(int lecturerId) {  
 this.lecturerid = lecturerId;  
 }  
  
 public String getLecturerName() {  
 return lecturer.getFirstname() + " " + lecturer.getLastname();  
 }  
  
 public Lecturer getLecturer() {  
 return lecturer;  
 }  
  
 public void setLecturer(Lecturer lecturer) {  
 this.lecturer = lecturer;  
 }  
  
 public int getWeekday() {  
 return weekday;  
 }  
  
 public void setWeekday(int weekDay) {  
 this.weekday = weekDay;  
 }  
  
 public int getLessonid() {  
 return lessonid;  
 }  
  
 public void setLessonid(int lessonId) {  
 this.lessonid = lessonId;  
 }  
  
 public Calendar getCalendar() {  
 return calendar;  
 }  
  
 public void setCalendar(Calendar calendar) {  
 this.calendar = calendar;  
 }  
  
}

***Main***

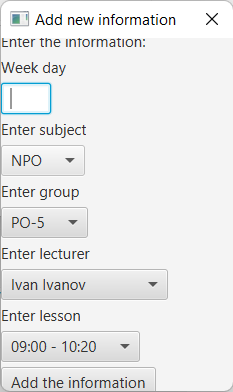
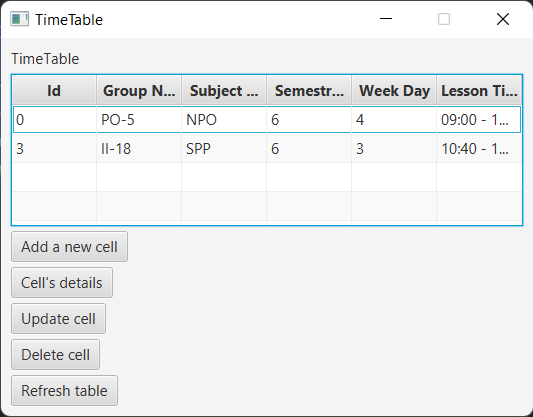
**App**

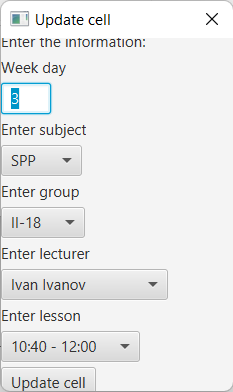
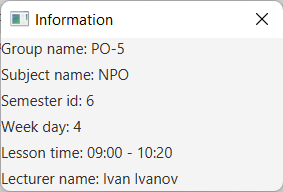
public class App extends Application {  
 public static DBManager *dbManager*;  
   
 public App() throws SQLException {  
 *dbManager* = new DBManager();  
 }  
   
 TableView<TimeTableClass> timeTableClass;  
   
 @Override  
 public void start(Stage primaryStage) throws Exception {  
 ArrayList<TimeTable> scheduleTables = *dbManager*.*scheduleTable*.GetAll();  
   
 ArrayList<TimeTable> scheduleTablesWithData = *convertWithData*(scheduleTables);  
   
 ArrayList<TimeTableClass> VmTimeTable = new ArrayList<>();  
   
 scheduleTablesWithData.forEach(t -> {  
 VmTimeTable.add(new TimeTableClass(t));  
 });  
  
 ObservableList<TimeTableClass> observableTimeTables = FXCollections.*observableArrayList*(VmTimeTable);  
  
 timeTableClass = new TableView<>(observableTimeTables);  
  
 timeTableClass.setColumnResizePolicy(TableView.*CONSTRAINED\_RESIZE\_POLICY*);  
 timeTableClass.setPrefSize(720,200);  
  
 TableColumn<TimeTableClass, Integer> idColumn = new TableColumn<>("Id");  
 idColumn.setCellValueFactory(new PropertyValueFactory<>("id"));  
 timeTableClass.getColumns().add(idColumn);  
  
 TableColumn<TimeTableClass, String> groupnameColumn = new TableColumn<>("Group Name");  
 groupnameColumn.setCellValueFactory(new PropertyValueFactory<>("groupname"));  
 timeTableClass.getColumns().add(groupnameColumn);  
  
 TableColumn<TimeTableClass, String> subjectNameColumn = new TableColumn<>("Subject Name");  
 subjectNameColumn.setCellValueFactory(new PropertyValueFactory<>("subjectName"));  
 timeTableClass.getColumns().add(subjectNameColumn);  
  
 TableColumn<TimeTableClass, Integer> semesteridColumn = new TableColumn<>("Semestre Id");  
 semesteridColumn.setCellValueFactory(new PropertyValueFactory<>("semesterid"));  
 timeTableClass.getColumns().add(semesteridColumn);  
  
 TableColumn<TimeTableClass, Integer> weekdayColumn = new TableColumn<>("Week Day");  
 weekdayColumn.setCellValueFactory(new PropertyValueFactory<>("weekday"));  
 timeTableClass.getColumns().add(weekdayColumn);  
  
 TableColumn<TimeTableClass, String> lessontimeColumn = new TableColumn<>("Lesson Time");  
 lessontimeColumn.setCellValueFactory(new PropertyValueFactory<>("lessontime"));  
 timeTableClass.getColumns().add(lessontimeColumn);  
  
 executeRefresh();  
  
 final VBox vbox = new VBox();  
 vbox.setSpacing(5);  
 vbox.setPadding(new Insets(10, 10, 10, 10));  
 vbox.getChildren().addAll(new Label("TimeTable"), timeTableClass);  
  
 Button addCellButton = new Button();  
 addCellButton.setText("Add a new cell");  
  
 addCellButton.setOnAction(event -> {  
 new TimeTableAddingDialog(primaryStage);  
 });  
  
 vbox.getChildren().addAll(addCellButton);  
  
  
 Button infoButton = new Button();  
 infoButton.setText("Cell's details");  
  
 infoButton.setOnAction(event -> {  
 TimeTableClass infoTimeTable = timeTableClass.getSelectionModel().getSelectedItems().get(0);  
 new InfoDialog(primaryStage, infoTimeTable.getGroupname(), infoTimeTable.getSubjectName(), infoTimeTable.getSemesterid(),  
 infoTimeTable.getWeekday(), infoTimeTable.getLessontime(), infoTimeTable.getLecturerFullName());  
 });  
  
 vbox.getChildren().addAll(infoButton);  
  
  
 Button updateButton = new Button();  
 updateButton.setText("Update cell");  
  
 updateButton.setOnAction(event -> {  
 TimeTableClass updateTimeTable = timeTableClass.getSelectionModel().getSelectedItems().get(0);  
 TimeTable vmtable = new TimeTable();  
  
 try {  
 vmtable = *dbManager*.*scheduleTable*.GetByIdOrNull(updateTimeTable.getId());  
 }  
 catch (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
  
 new TimeTableUpdatingDialog(primaryStage, vmtable);  
 });  
  
 vbox.getChildren().addAll(updateButton);  
  
  
 Button deleteCellButton = new Button();  
 deleteCellButton.setText("Delete cell");  
  
 deleteCellButton.setOnAction(event -> {  
 try {  
 executeDelete();  
 executeRefresh();  
 }  
 catch (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 });  
  
 vbox.getChildren().addAll(deleteCellButton);  
  
  
 Button refreshCellButton = new Button();  
 refreshCellButton.setText("Refresh table");  
  
 refreshCellButton.setOnAction(event -> {  
 try {  
 executeRefresh();  
 }  
 catch (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 });  
 vbox.getChildren().addAll(refreshCellButton);  
  
  
 Parent root = FXMLLoader.*load*(getClass().getResource("sample.fxml"));  
 primaryStage.setTitle("TimeTable");  
  
 Scene scene = new Scene(root, 500, 500);  
 ((GridPane) scene.getRoot()).getChildren().addAll(vbox);  
 primaryStage.setScene(scene);  
  
 primaryStage.setHeight(425);  
 primaryStage.setWidth(550);  
  
 primaryStage.setResizable(false);  
  
 primaryStage.show();  
 }  
  
 public static void main(String[] args) {  
 *launch*(args);  
 }  
  
 private void executeRefresh() throws SQLException {  
 timeTableClass.getItems().clear();  
  
 ArrayList timeTables = *dbManager*.*scheduleTable*.GetAll();  
  
 ArrayList timeTablesWithData = *convertWithData*(timeTables);  
 ArrayList<TimeTableClass> VmTimeTable = new ArrayList<>();  
  
 timeTablesWithData.forEach(t -> {  
 VmTimeTable.add(new TimeTableClass((TimeTable) t));  
 });  
  
 ObservableList<TimeTableClass> observableTimeTables = (ObservableList<TimeTableClass>) FXCollections.*observableArrayList*(VmTimeTable);  
  
 timeTableClass.getItems().addAll(observableTimeTables);  
 }  
  
 private void executeDelete() throws SQLException {  
 TimeTableClass deletedTimeTable = timeTableClass.getSelectionModel().getSelectedItems().get(0);  
 *dbManager*.*scheduleTable*.Delete(new TimeTable(deletedTimeTable.getId()));  
 }  
  
 public static ArrayList<TimeTable> convertWithData(ArrayList<TimeTable> table) {  
 table.forEach(t -> {  
 try {  
 t.setGroup(DBManager.*groupTable*.GetByIdOrNull(t.getGroupid()));  
 t.setSubject(DBManager.*subjectTable*.GetByIdOrNull(t.getSubjectid()));  
 t.setLecturer(DBManager.*lecturerTable*.GetByIdOrNull(t.getLecturerid()));  
 t.setCalendar(DBManager.*calendarTable*.GetByIdOrNull(t.getLessonid()));  
 }  
 catch (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 });  
  
 return table;  
 }  
}

**TimeTableClass**

public class TimeTableClass {  
 private int id;  
  
 private String groupname;  
  
 private String subjectName;  
  
 private String lecturerFullName;  
  
 private int semesterid;  
 private int weekday;  
 private String lessontime;  
  
 private int groupid;  
 private int subjectid;  
 private int lecturerid;  
 private int lessonid;  
  
 public TimeTableClass(TimeTable table) {  
 this.id = table.getId();  
 this.groupname = table.getGroup().getGroupname();  
 this.subjectName = table.getSubject().getSubjectName();  
 this.semesterid = table.getCalendar().getSemesterid();  
 this.weekday = table.getWeekday();  
 this.lessontime = table.getCalendar().getLessontime();  
 this.groupid = table.getGroupid();  
 this.subjectid = table.getSubjectid();  
 this.lessonid = table.getLessonid();  
 this.lecturerFullName = table.getLecturerName();  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getGroupname() {  
 return groupname;  
 }  
  
 public void setGroupname(String groupname) {  
 this.groupname = groupname;  
 }  
  
 public String getSubjectName() {  
 return subjectName;  
 }  
  
 public void setSubjectName(String subjectName) {  
 this.subjectName = subjectName;  
 }  
  
 public int getSemesterid() {  
 return semesterid;  
 }  
  
 public void setSemesterid(int semesterid) {  
 this.semesterid = semesterid;  
 }  
  
 public int getWeekday() {  
 return weekday;  
 }  
  
 public void setWeekday(int weekday) {  
 this.weekday = weekday;  
 }  
  
 public int getLessonid() {  
 return lessonid;  
 }  
  
 public void setLessonid(int lessonid) {  
 this.lessonid = lessonid;  
 }  
  
 public String getLessontime() {  
 return lessontime;  
 }  
  
 public void setLessontime(String lessontime) {  
 this.lessontime = lessontime;  
 }  
  
 public String getLecturerFullName() {  
 return lecturerFullName;  
 }  
}

**Результаты:**

****

****

**Вывод:** приобрел практические навыки разработки многооконных приложений на JavaFX для работыс базами данных.